The IMAP WEBPUSH Extension

Simon Gougeon – UnifiedPush
2025-03-20 – IETF122
https://datatracker.ietf.org/doc/draft-gougeon-imap-webpush/

Introduction

Introduction

Push notifications = Short messages sent by the server to a disconnected client

WebPush = RFC8030 (HTTP Push) + RFC8291 (encryption) + RFC8292 (VAPID – authorization)

WebPush defines a way for applications to **deliver real-time events** in a timely fashion, with push notifications

Push notifications allows **consolidating all real-time events into a single session** which ensures more efficient use of network and radio resources. They are **particularly used in mobile environments**.

Push notifications may also be used for other use cases, for example during the migration to a new server, the new server may subscribe to the old server to be notified when an event is recorded.

Introduction

Many use cases have led to a **need for real-time events with email**. IMAP support for real-time events has been added with the IDLE command and the NOTIFY extension. These commands require using a persistent connection per account and contribute to unnecessary use of the device radio.

JMAP has responded to this need by **supporting WebPush from the beginning**.

Proprietary softwares have implemented their own API.

Therefore, this extension permits IMAP servers to send WebPush notifications.

Introduction

Known interested project:

- **Mox** – modern, secure, all-in-one email server:
  - https://www.xmox.nl/
  - POC of the draft already implemented

- **DeltaChat** – Instant messaging app built on emails:
  - https://delta.chat

May interest:

- Other email servers
- Webmail clients
- Mobile client:
  - Google Push services (FCM) can be requested with WebPush
  - UnifiedPush uses WebPush
  - WebPush-to-APNS (Apple services) services are common

Introduction

Related work:

- **COI** (Chat Over Imap, discontinued) – WEBPUSH extension
  - Use METADATA extension
  - https://github.com/coi-dev/coi-specs/blob/master/webpush-spec.md
- Unspecified XAPPLEPUSHSERVICE extension – to APNS
  - https://github.com/st3fan/dovecot-xaps-plugin
- **IRCv3** – WebPush extension – Specifications close to the IMAP WEBPUSH extension
  - https://github.com/ircv3/ircv3-specifications/pull/471

Conventions

In examples, "**C:**" and "**S:**" indicate lines sent by the client and server, respectively.

"Push:" indicates lines send by the server with a push message

"**Push>**" indicates lines that are part of the same push message than the previous line

Lines ending in "\\" are interrupted for presentation reasons, they would actually be joined to the next line.

Note that each other line includes the terminating CRLF.

Design choices

- **Avoid unnecessary complexity**
  - Ignoring a push message uses insignificant resources, it’s OK to get some event we ignore (up to tens a day)
- If possible, **allow clients to get important information** (about incoming emails) without having to synchronize with the server
- Allow easy server-side implementation of **a generic push event** to enable specification-compliant “synchronization on push”
- Allow easy **client-side implementation of “(partial) synchronization on push”**, regardless of whether the server implements the generic push event or sends full events.
- **Do not depend on another extension**, especially for sync-on-push
  - 2 responses come from other extensions (UIDFETCH, VANISHED), but the full extensions don’t have to be supported.
- Push notifications alone don’t have to allow a fully synchronized state with the server
  - Delivery not guaranteed, out of order
- **Use VAPID** by default
  - Some push servers require it

Overview

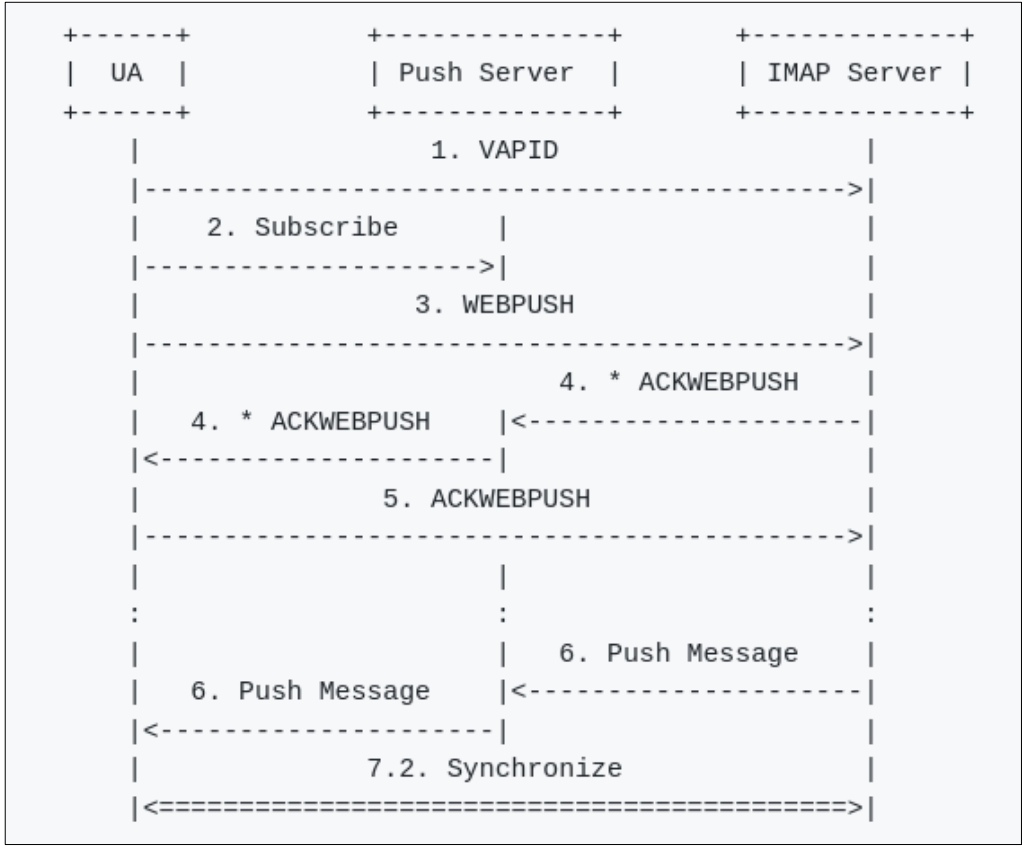
This extension adds 5 commands:

- **GETVAPID**:
  - Get the server VAPID public key
- **WEBPUSH**
  - Subscribe a new push subscription (endpoints+keys)
- **ACKWEBPUSH**
  - Acknowledge a subscription
  - Required to get push notifications, to avoid abuse
- **LWEBPUSH**
  - List current subscriptions
- **SILWEBPUSH**
  - Silent a subscription, for a given time or until a connection ends

Lifecycle of a subscription

Lifecycle of a subscription

1. Client gets Server VAPID key
2. Client registers to its push server with the VAPID key
3. Client subscribes new endpoint, encryption key
4. Server sends a verification token
5. Client verifies the push subscription
6. Server sends a push notification
7. Client handles the event
    1. Parses the content
    2. Synchronizes
8. Client may delete a subscription



1. Client gets Server VAPID key

C: a1 GETVAPID

S: * VAPID \
BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: a1 OK GETVAPID completed

2. Client registers to its push server

with the VAPID key

Example from MDN

JS

```
this.onpush = (event) => {
  console.log(event.data);
  // From here we can write the data to IndexedDB, send it to any open
  // windows, display a notification, etc.
};

navigator.serviceWorker.register("serviceworker.js");

// Use serviceWorker.ready to ensure that you can subscribe for push
navigator.serviceWorker.ready.then((serviceWorkerRegistration) => {
  const options = {
    userVisibleOnly: true,
    applicationServerKey,
  };
  serviceWorkerRegistration.pushManager.subscribe(options).then(
    (pushSubscription) => {
      console.log(pushSubscription.endpoint);
      // The push subscription details needed by the application
      // server are now available, and can be sent to it using,
      // for example, the fetch() API.
    },
    (error) => {
      // During development it often helps to log errors to the
      // console. In a production environment it might make sense to
```

3. Client subscribes new endpoint

and encryption key

C: a2 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
https://push.example.net/push/random1 \
BCVxsr7N_eNgVRqvHtD0zTZsEc6-VV-JvLexhqUzORcxaOzi6-AYWXvTBHm4bj\
yPjs7Vd8pZGH6SRpkNtoIAiw4 \
BTBZMqHH6r4Tts7J_aSIgg

S: * VAPID BOniQ9xHBPNY9gnQW4o-16vHqOb40pEIMifyUdFsxAgy\
zVkFMguxw0QrdbZcq8hRjN2zpeInRvKVPlkzABvuTnI
S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
https://push.example.net/push/random1 NIL
S: a2 OK WEBPUSH completed

4. Server sends a verification token

Push: * ACKWEBPUSH **585078c5-fb8b-4ed0-8e77-474ab08f0a30**

5. Client verifies the push subscription

C: a2 ACKWEBPUSH **585078c5-fb8b-4ed0-8e77-474ab08f0a30**

S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
https://push.example.net/push/JzLQ3raZJfFBR0aqvOMsLrt54w4rJUsV 0
S: a2 OK ACKWEBPUSH completed

6. Server sends a push notification

Push: * SELECT "New Messages"
Push> * 3 UIDFETCH (ENVELOPE \
("Mon, 7 Feb 1994 21:52:25 -0800 (PST)" \
"afternoon meeting" \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
((NIL NIL "mooch" "owatagu.siam.edu.example")) \
NIL NIL NIL "<B27397-0100000@Blurdybloop.example>"))

7.1. Client parses the content

Push: * SELECT "New Messages"
Push> * 3 UIDFETCH (ENVELOPE \
("Mon, 7 Feb 1994 21:52:25 -0800 (PST)" \
"afternoon meeting" \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
(("Fred Foobar" NIL "foobar" "Blurdybloop.example")) \
((NIL NIL "mooch" "owatagu.siam.edu.example")) \
NIL NIL NIL "<**B27397-0100000@Blurdybloop.example**>"))

7.2. Client synchronizes with the server

Push: * SELECT **"New Messages"**
Push> * 3 UIDFETCH (ENVELOPE \
("Mon, 7 Feb 1994 21:52:25 -0800 (PST)" \
"afternoon meeting" \
(("Fred Foobar"
(("Fred Foobar"
(("Fred Foobar"
((NIL NIL "mooc
NIL NIL NIL "<B

C: a3 **SELECT "New Messages"**

S: * 172 EXISTS
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 4392] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen *)] Limited
S: * LIST () "/" INBOX
S: A142 OK [READ-WRITE] SELECT completed

8. Client may delete a subscription

C: a4 LWEBPUSH *

S: * WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 \
https://push.example.net/push/JzLQ3raZJfFBR0aqvOMsLrt54w4rJUsV NIL
S: a4 OK LWEBPUSH completed

C: a5 WEBPUSH a8282bf9-6102-4e1b-bb61-d26d0e532e65 **NIL**

S: a5 OK WEBPUSH completed
C: a6 LWEBPUSH *
S: a6 OK LWEBPUSH completed

Other cases handled in the draft

- New capability
- VAPID key rotation
- Silencing a subscription
- Listing subscriptions
- WebPush headers
- Push server response
- Response too long for a push message
- Multiple responses in a single push message

Security & IANA Considerations

Security Considerations

- The privacy and security considerations of WebPush all apply to the use of this extension.

- WebPush on decentralized applications **may be used as a DDOS amplification:**
  - by registering multiple time a target as the endpoint, on multiple servers then notifying all the accounts
  - **Requiring the client to acknowledge one push notification** (with the ACKWEBPUSH command and response) **greatly reduces this risk.**

IANA Considerations

- WEBPUSH capability need to be registered.

Pending questions

Pending questions

- Allow client to try working with push notifications only, with a counter associated to each push notifications to detect out of order/undelivered events ?
  - *Push> PUSH push-counte*r
  - *Push: SELECT INBOX*
  - *Push: [...]*
- Should we add a way for the client to subscribe for SYNC responses only ? To select only some responses ?
  - cf. design choices: avoid complexity, some ignored push notifications is OK
- Select a compression identifier ?
- Formal syntax
  - Atoms ?
  - Endpoint syntax has to be fixed

Questions ?

Thanks to Mechiel Lukkien (mox) for the review of the draft, and the review of the presentation,

Thanks to NLNet for the grant to work on this draft.